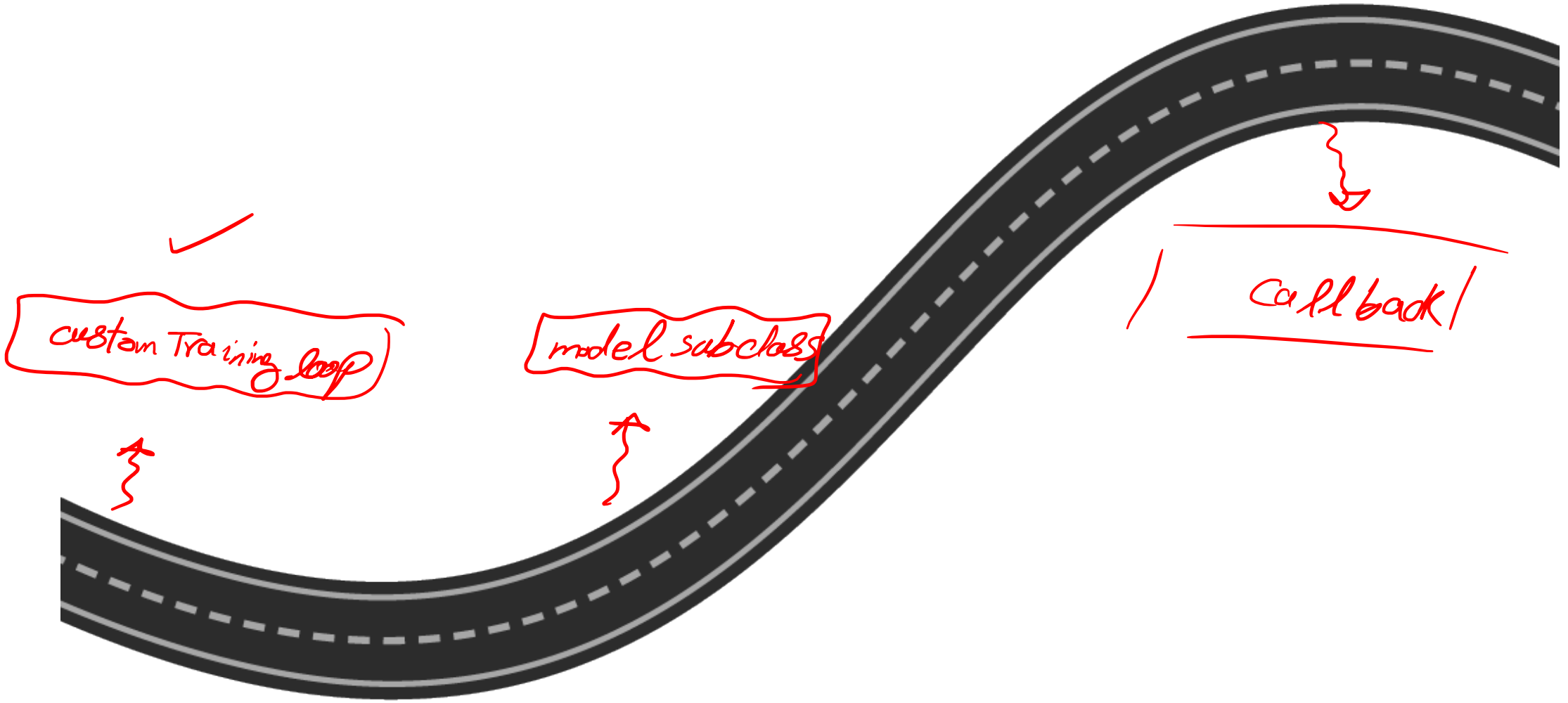
آکادمی ربوتک

دوره تنسورفلو پیشرفته

جلسه هفتم : پیاده سازی با Model Subclassing و Callbacks

چه گفته ایم و چه خواهیم گفت:



ای نسیم سحر آرامگه یار کجاست
منزل آن مه عاشق‌کش عیار کجاست

شب تار است و ره وادی ایمن در پیش
آتش طور کجا موعد دیدار کجاست

آن کس است اهل بشارت که اشارت داند
نکته‌ها هست بسی محرم اسرار کجاست

ساقی و مطرب و می جمله مهیاست ولی
عیش بی یار مهیا نشود یار کجاست

حافظ از باد خزان در چمن دهر مرنج
فکر معقول بفرما گل بی خار کجاست

مروری بر کد جلسه پیش

نمودار Loss ؟

امین بمانی : نمودار Loss داده های Test هم میخواهیم.

```
def perform_validation():
    val_loss = []
    for i in range(0, bat_per_epoch_test):
        start = i* batch_size
        end = start + batch_size
        y_pred = model(x_test[start:end])
        loss_on_test_data = categorical_crossentropy(y_test[start:end], y_pred)
        val_loss.append(loss_on_test_data)

    return np.mean(val_loss)
```

Metric ها چطور ؟

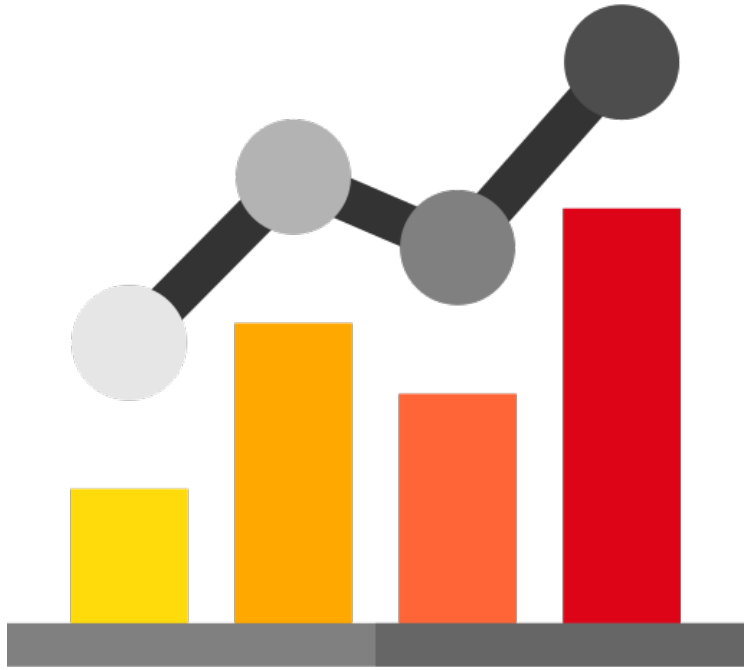
ابتدا یک مثال ساده:

```
from tensorflow.keras import metrics


m = metrics.CategoricalAccuracy()
y_true = [[1, 0, 0], [0, 0, 1]]
y_pred = [[0.1, 0.8, 0.1], [0.1, 0.1, 0.8]]
m.update_state(y_true, y_pred)


print("acc:", m.result().numpy())
```

reset متد

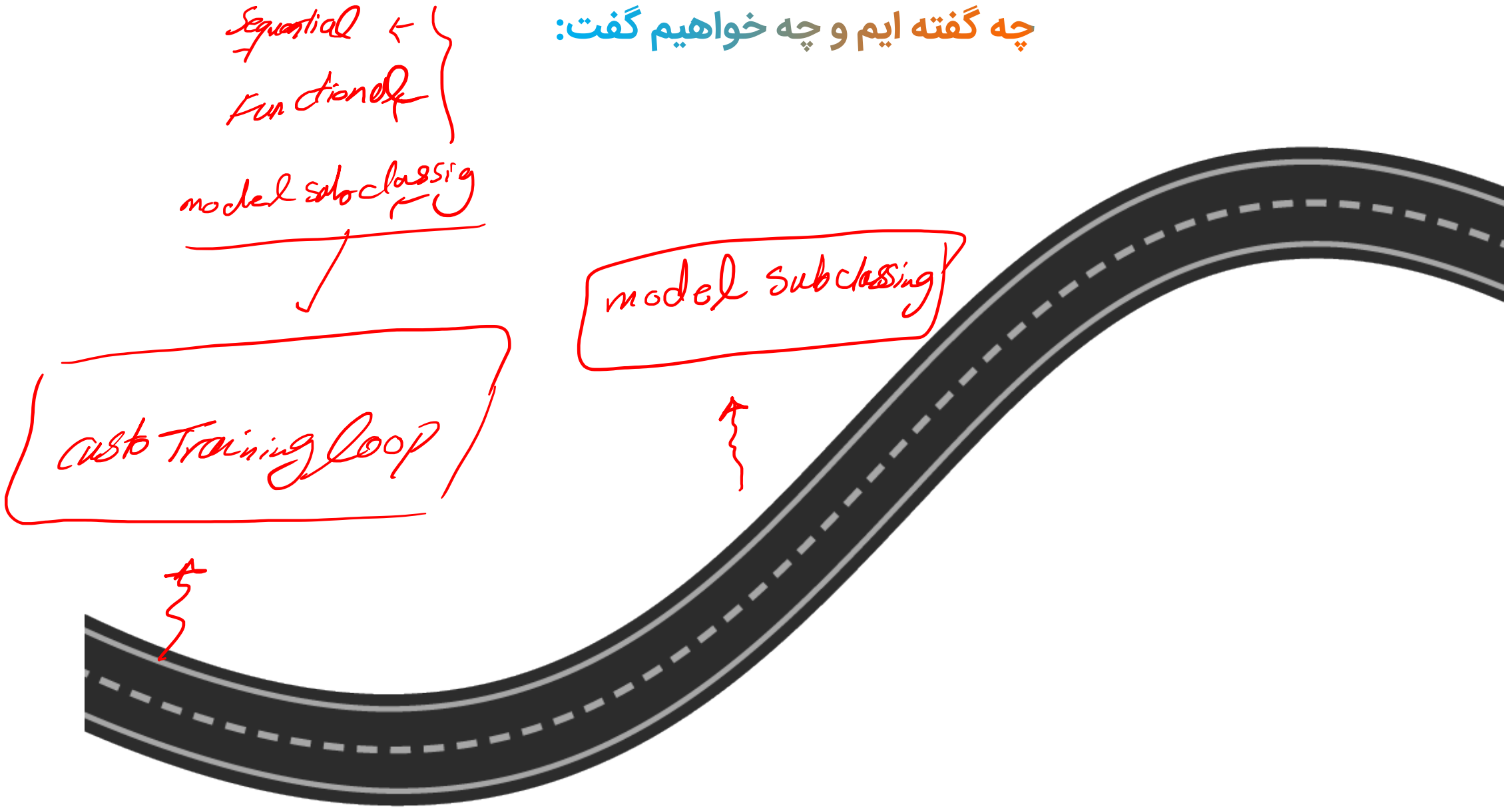
```
m.reset_state()
print("acc:", m.result().numpy())
```

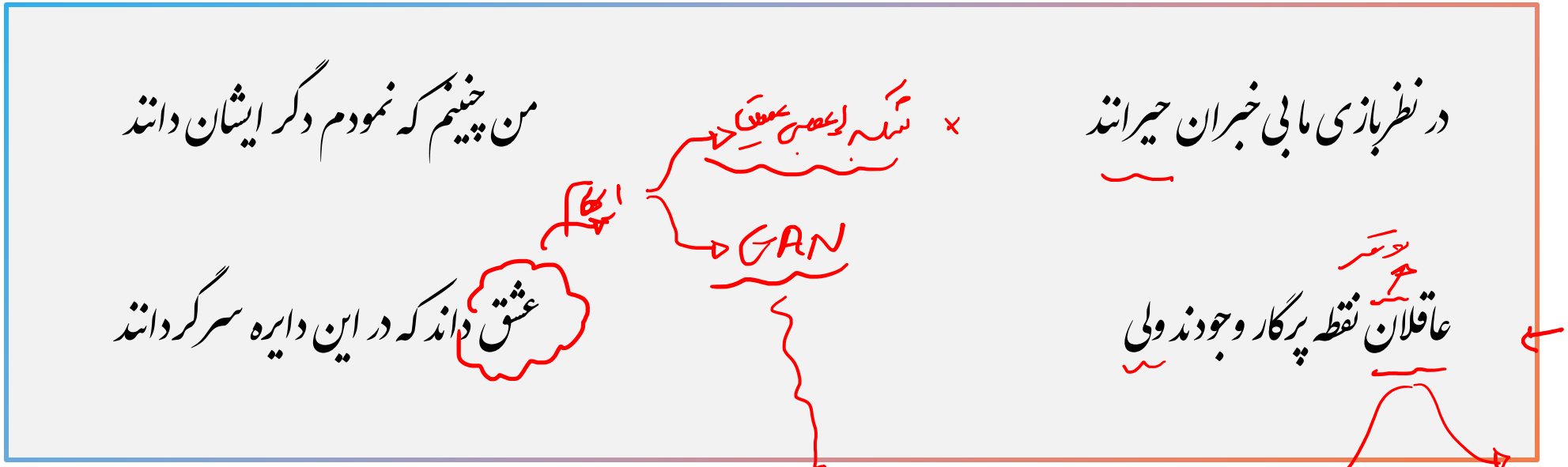
همین ها رو در کد پیاده سازی کنیم !



```
train_metric = metrics.CategoricalAccuracy()
test_metric = metrics.CategoricalAccuracy()

train_metric.update_state(y_train[start:end], y_pred)
test_metric.update_state(y_test[start:end], y_pred)

train_acc = train_metric.result().numpy()
test_acc = test_metric.result().numpy()
```

چه گفته ایم و چه خواهیم گفت:




در نظربازی ما بی خبران حیرانند
من چنینم که نمودم دگر ایشان دانند
عاقلان نقطه پرگار وجودند ولی
عشق داند که در این دایره سرگردانند
Sequential
Functional
GAN

Model Subclassing

class — (Model)

def — init —

def call

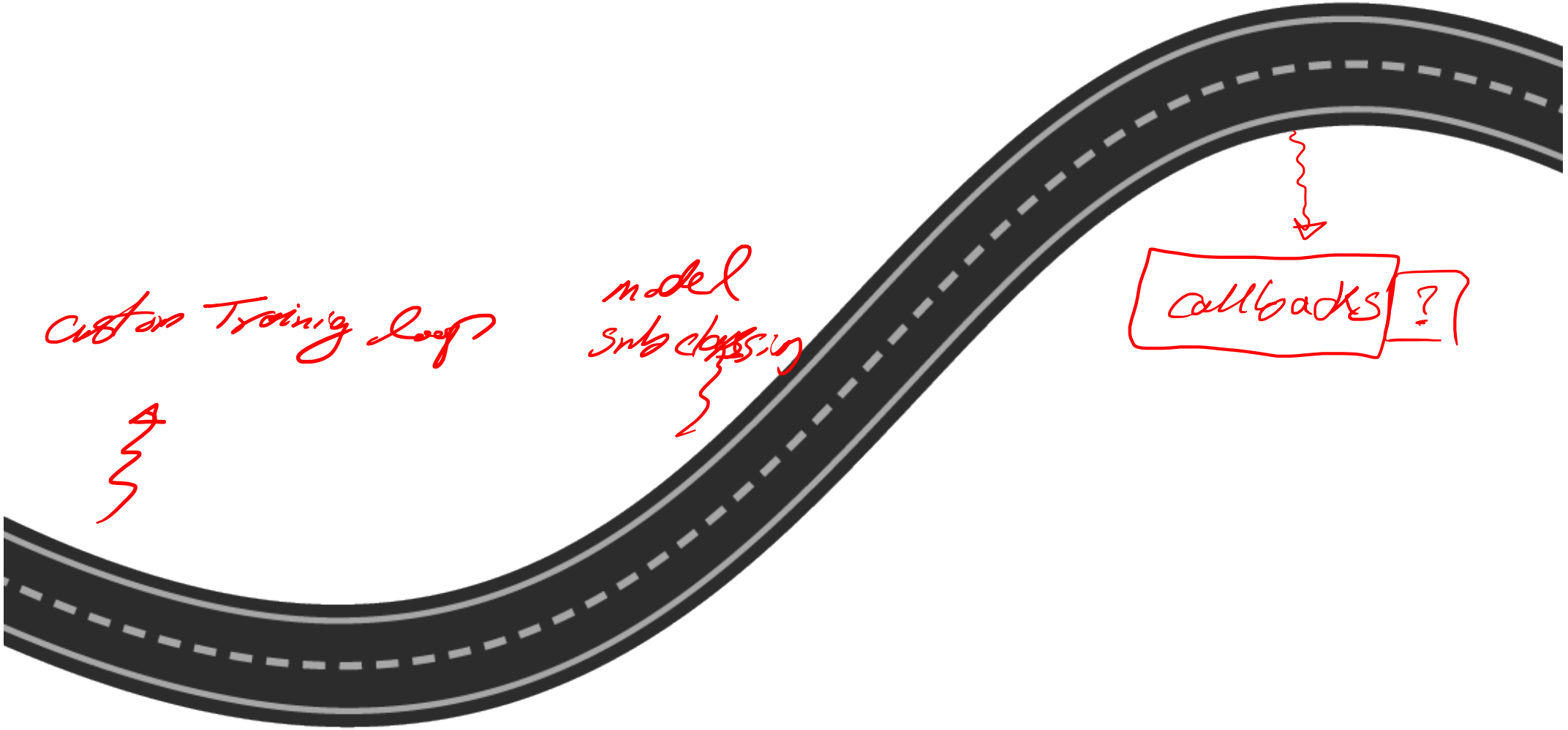
متد init

```
class GNet(models.Model):
    def __init__(self):
        super().__init__()
        self.conv1 = layers.Conv2D(32, (3, 3), activation='relu')
        self.conv2 = layers.Conv2D(64, (3, 3), activation='relu')
        self.pool1 = layers.MaxPooling2D((2, 2))
        self.drop1 = layers.Dropout(0.25)
        self.flat = layers.Flatten()
        self.dense1 = layers.Dense(128, activation='relu')
        self.drop2 = layers.Dropout(0.5)
        self.dense2 = layers.Dense(10, activation="softmax")
```

متد call

```
def call(self, inputs):
    x = self.conv1(inputs)
    x = self.conv2(x)
    x = self.pool1(x)
    x = self.drop1(x)
    x = self.flat(x)
    x = self.dense1(x)
    x = self.drop2(x)
    x = self.dense2(x)

    return x
```

چه گفته ایم و چه خواهیم گفت:



Callback کلا چیست ؟

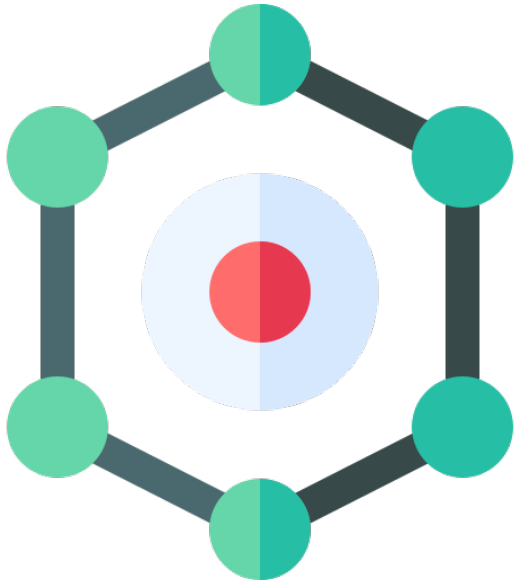
کالبک یک سری object هستند که فرآیند Training را انعطاف پذیرتر و تحلیل خروجی را ساده تر می کنند.

کالبک هایی که خواهیم گفت:

- EarlyStopping
- CSVLogger
- ModelCheckPoint
- LabmdaCallback
- BaseLoggeer
- TensorBoard
- TerminateOnNan
- History
- LearningRateScheduler
- ReduceLROnPlateau

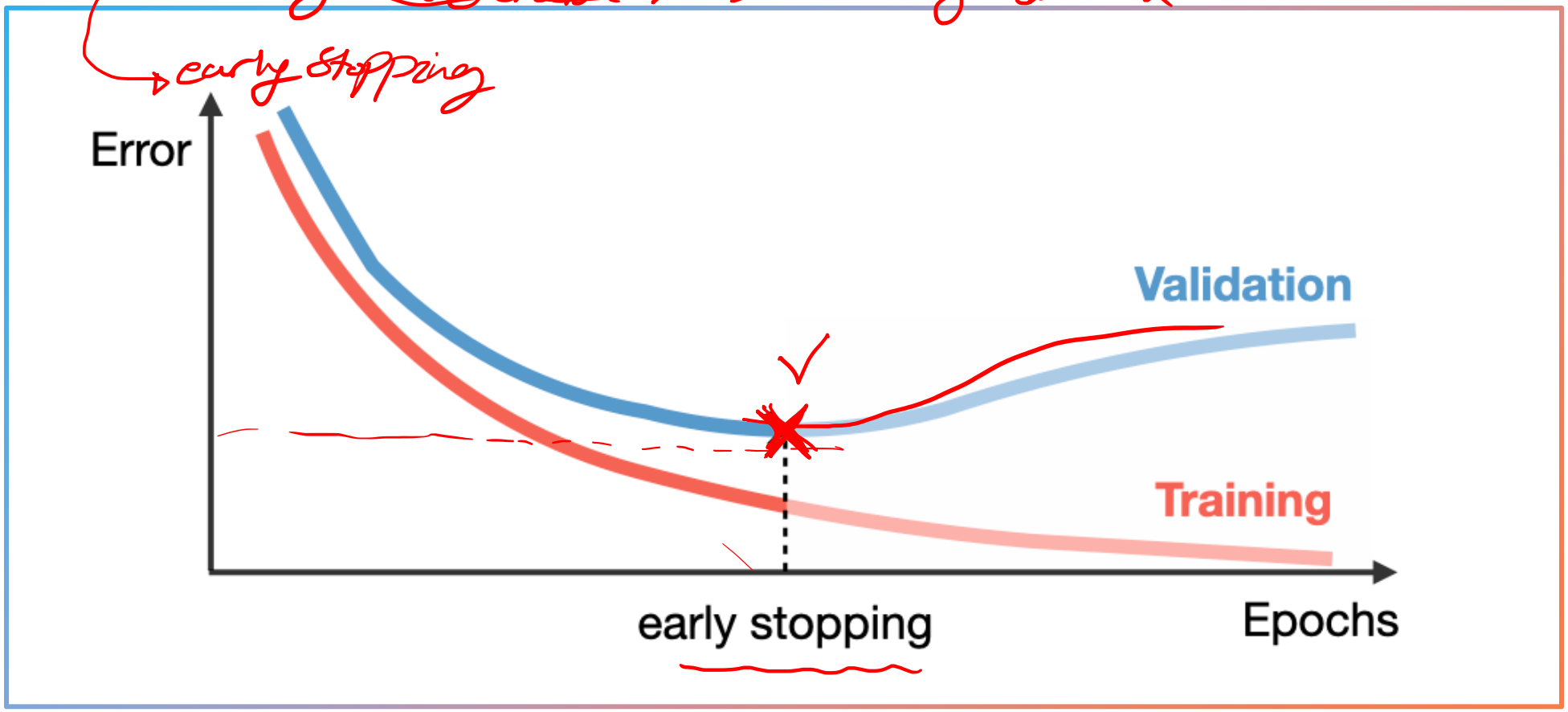
ساختار کلی استفاده از کالبک ها:

```
callback_object = tf.keras.callback.XCallBack()

model.fit(X, y, callbacks=[callback_object])
```



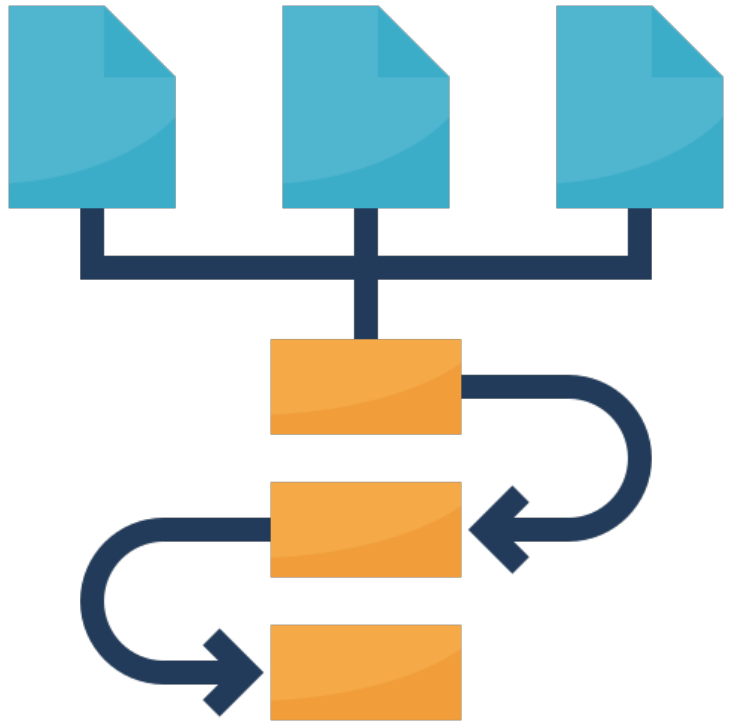
overfitting → Regularization
→ Dropout
→ ImageDataGenerator ~ Data augmentation
→ early Stopping

Early Stopping



گام ۱: فراخوانی

```
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='auto'
)
```

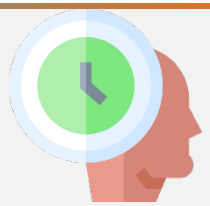


پارامترهای مهم

monitor

min_delta

patience

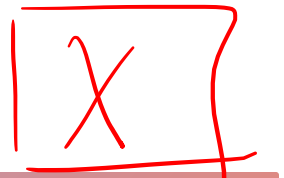


mode

گام ۲ : استفاده

```
history = model.fit(
    X_train,
    y_train,
    epochs=50,
    validation_split=0.20,
    batch_size=64,
    verbose=2,
    callbacks=[early_stopping]
)
plot_metric(history, 'loss')
```

Result:

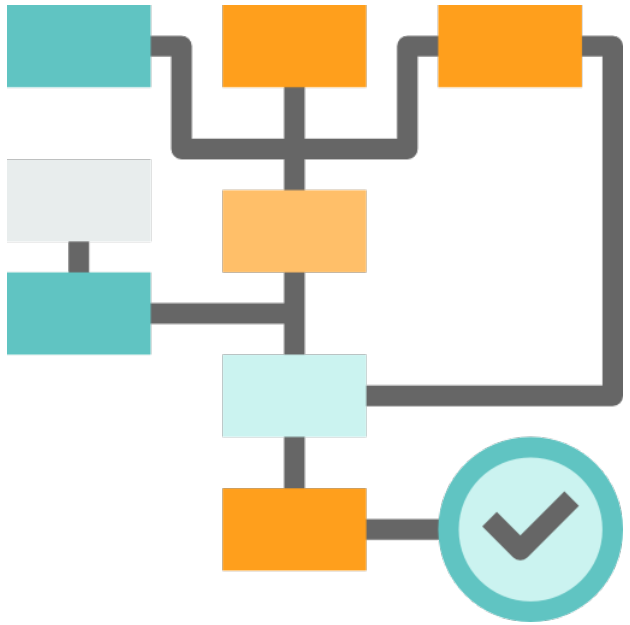


```
125/125 - 0s - loss: 0.3920 - accuracy: 0.8637 - val_loss: 0.4520 - val_accuracy: 0.8480
Epoch 32/50
125/125 - 0s - loss: 0.3864 - accuracy: 0.8691 - val_loss: 0.4560 - val_accuracy: 0.8405
Epoch 33/50
125/125 - 0s - loss: 0.3817 - accuracy: 0.8686 - val_loss: 0.4568 - val_accuracy: 0.8440
Epoch 34/50
125/125 - 0s - loss: 0.3789 - accuracy: 0.8735 - val_loss: 0.4523 - val_accuracy: 0.8425
Epoch 35/50
125/125 - 0s - loss: 0.3754 - accuracy: 0.8710 - val_loss: 0.4458 - val_accuracy: 0.8465
Epoch 36/50
125/125 - 0s - loss: 0.3710 - accuracy: 0.8735 - val_loss: 0.4637 - val_accuracy: 0.8395
Epoch 37/50
125/125 - 0s - loss: 0.3678 - accuracy: 0.8759 - val_loss: 0.4844 - val_accuracy: 0.8295
Epoch 38/50
125/125 - 0s - loss: 0.3641 - accuracy: 0.8766 - val_loss: 0.4463 - val_accuracy: 0.8450
Epoch 39/50
125/125 - 0s - loss: 0.3595 - accuracy: 0.8766 - val_loss: 0.4549 - val_accuracy: 0.8360
```

CSVLogger

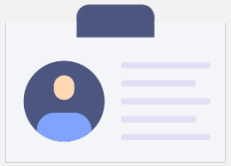
گام ۱: فراخوانی

```
from tensorflow.keras.callbacks import CSVLogger

csv_log = CSVLogger("results.csv", separator=',', append=False)
```

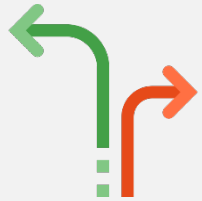


پارامترهای مهم

Filename



Separator



append

گام ۲ : استفاده

```
history = model.fit(
    X_train,
    y_train,
    epochs=50,
    validation_split=0.20,
    batch_size=64,
    verbose=2,
    callbacks=[csv_log]
)
plot_metric(history, 'loss')
```

Result:

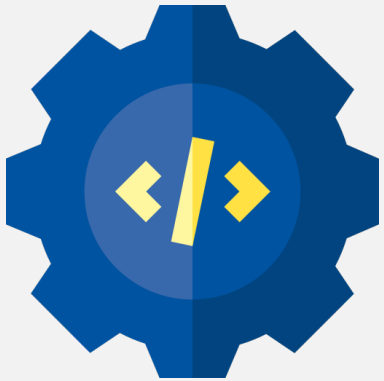
```
epoch,accuracy,loss,val_accuracy,val_loss
0,0.5173749923706055,1.605967879295349,0.656499981880188,1.1670798063278198
1,0.6796249747276306,0.9883725047111511,0.6940000057220459,0.898496270179748
2,0.7222499847412109,0.8203113675117493,0.7214999794960022,0.798286259174346
3,0.7494999766349792,0.7392875552177429,0.7509999871253967,0.735735595226287
4,0.765625,0.6863712668418884,0.7549999952316284,0.699014663696289
5,0.7814999818801880,0.6473094820976257,0.7735000252723694,0.658950746059417
6,0.7891250252723694,0.6175520420074463,0.7620000243186951,0.661935448646545
7,0.8009999990463257,0.5934941172599792,0.7904999852180481,0.615208327770233
8,0.8068749904632568,0.5734546184539795,0.7904999852180481,0.603227913379669
```

ModelCheckPoint

انواع روش های ذخیره سازی مدل

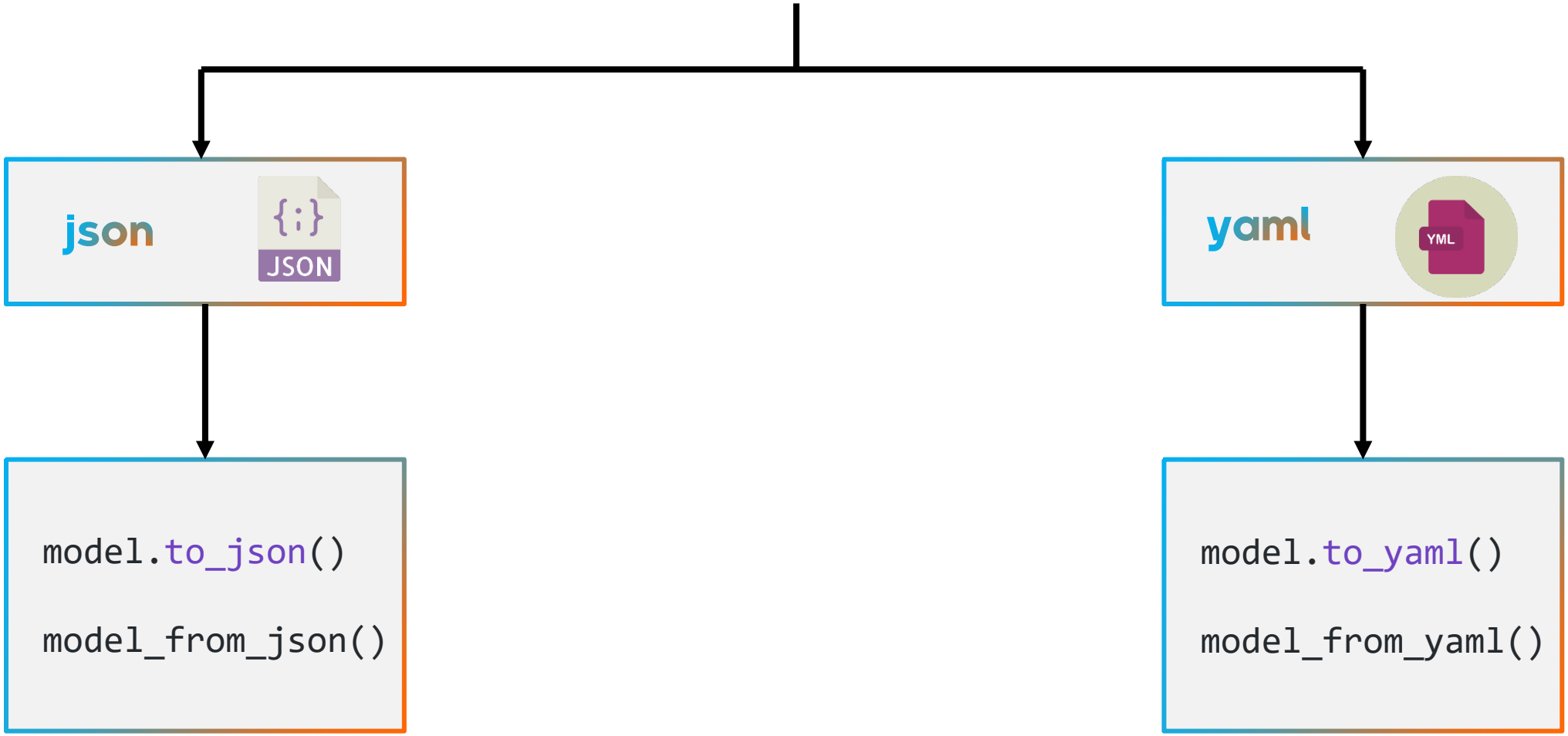
Model Weights

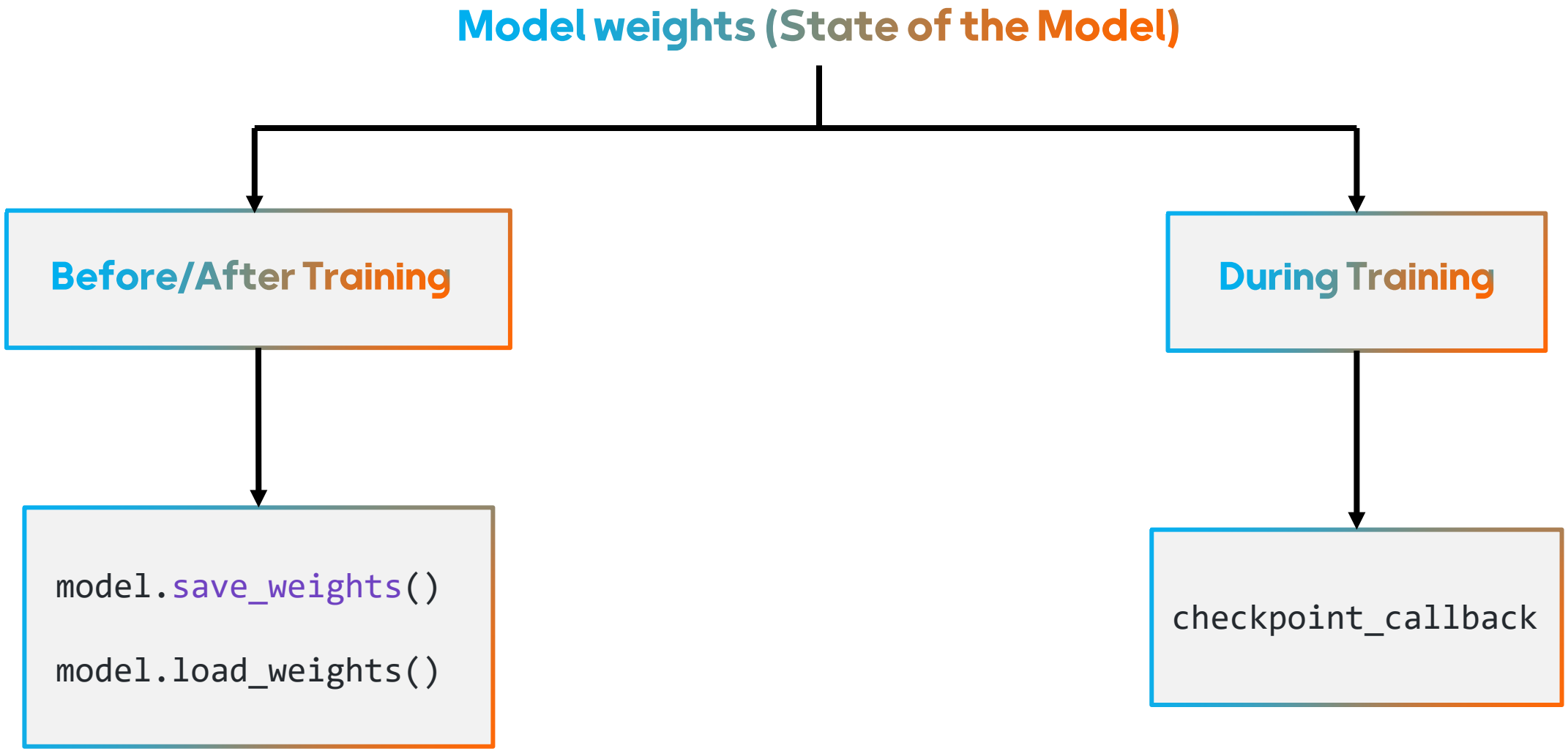
Model Architecture

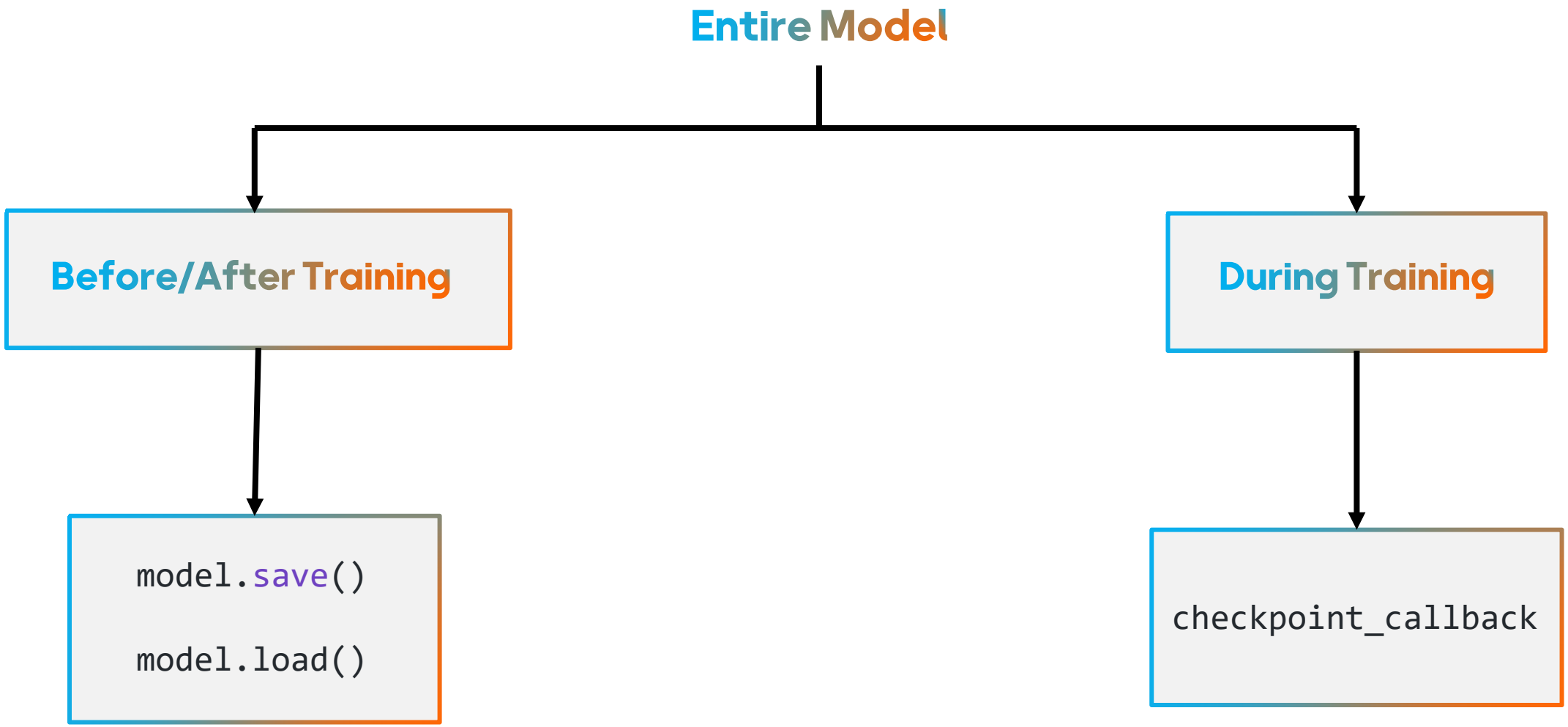


Entire Model

Model Architecture (Layers and Connections)

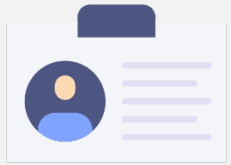



Model weights (State of the Model)
Before/After Training
During Training
model.save_weights()
model.load_weights()
checkpoint_callback


Entire Model
Before/After Training
During Training
model.save()
model.load()
checkpoint_callback

پارامترهای ورودی

FilePath



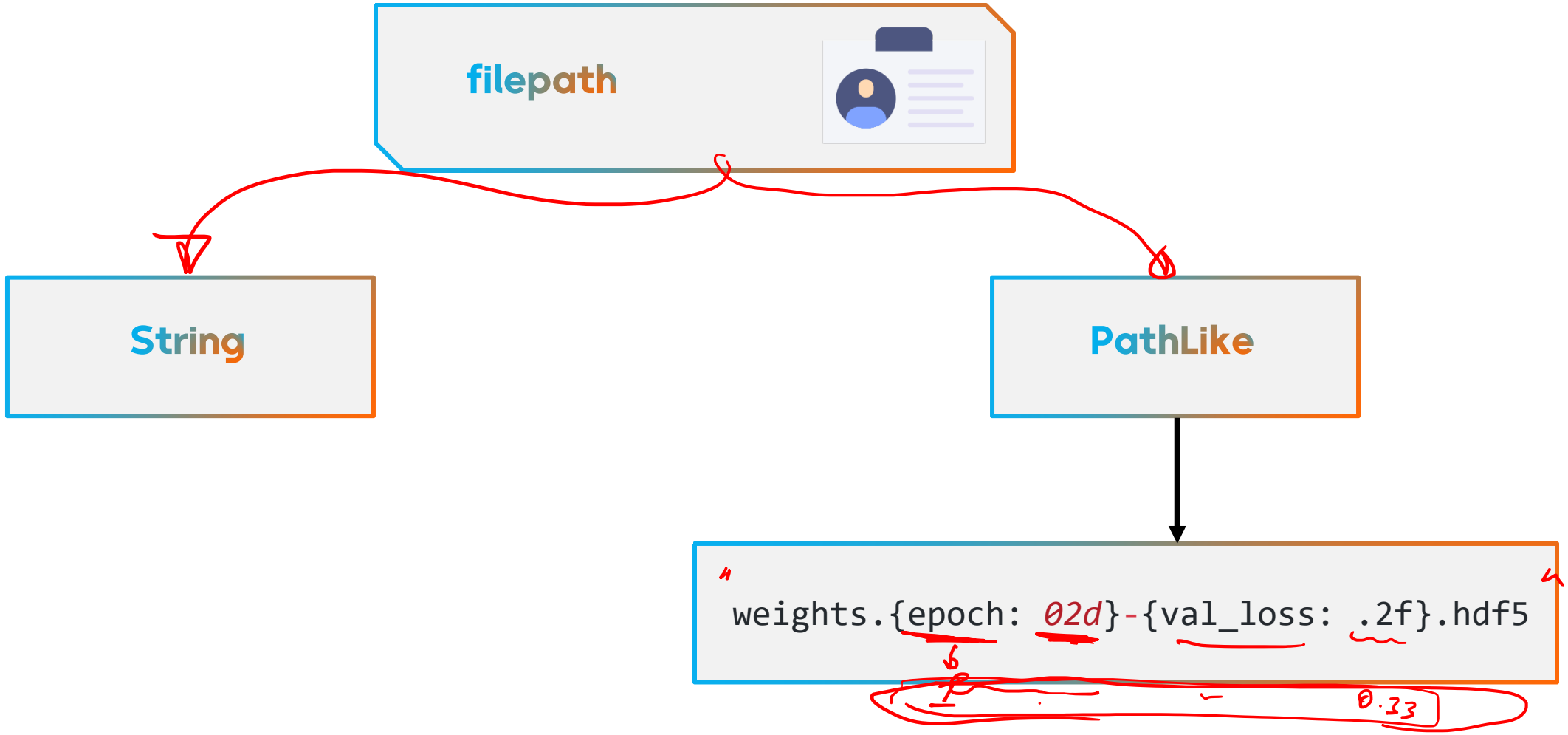
save_best_only

monitor

mode

save_weights_only

save_freq


filepath
String
PathLike
weights.{epoch: 02d}-{val_loss: .2f}.hdf5

در کد تست کنیم !

```
checkpoint_path = 'model_checkpoints/model.{epoch: 02d}-{val_loss: .2f}.hdf5'

checkpoint = ModelCheckpoint(
    filepath=checkpoint_path,
    #save_best_only=True
)
```

فقط وزن ها !

```
checkpoint_path = 'model_checkpoints/weight.{epoch: 02d}-{val_loss: .2f}.hdf5'

checkpoint = ModelCheckpoint(
    filepath=checkpoint_path,
    save_best_only=True,
    save_weights_only=True
)
```

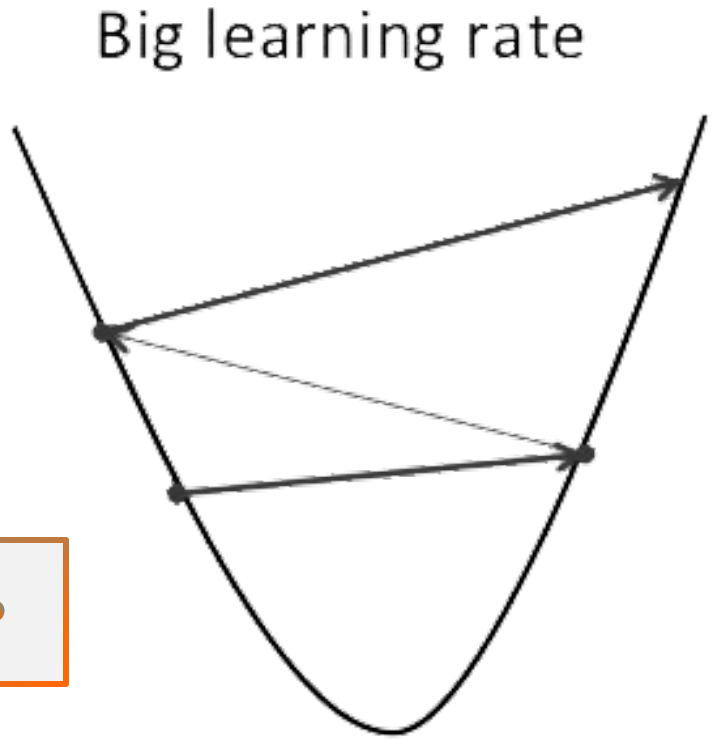
من مست و تو دیوانه ما را کی برد خانه

من چند تو را گفتم کم خور دو سه پیمانه

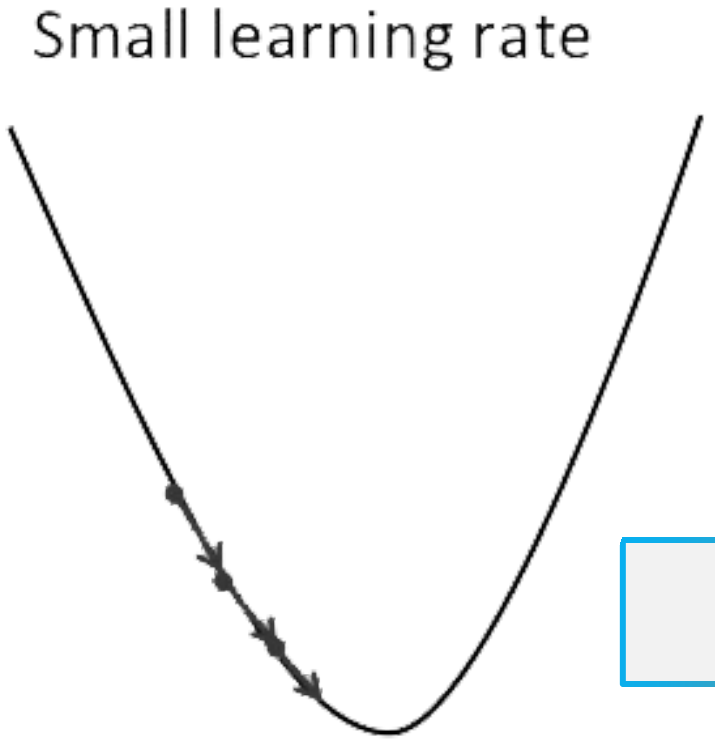
در شهر یکی کس را هشیار نمی‌بینم

هر یک بتر از دیگر شوریده و دیوانه

به نظر منظور از مولانا واگرایی شبکه عصبی در اثر تنظیم بد هایپرپارامترها می باشد.



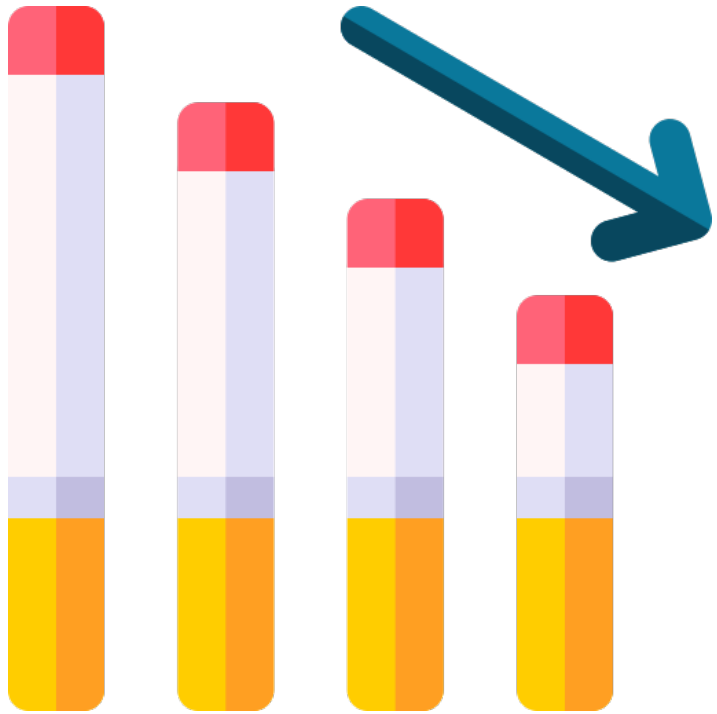
هست و دیوانه



عاقل و بالغ

ReduceLROnPlateau

کاهش نرخ LR پس از عدم بهبودی متریک های اندازه گیری

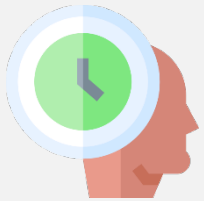


پارامترها

monitor

min_delta

patience



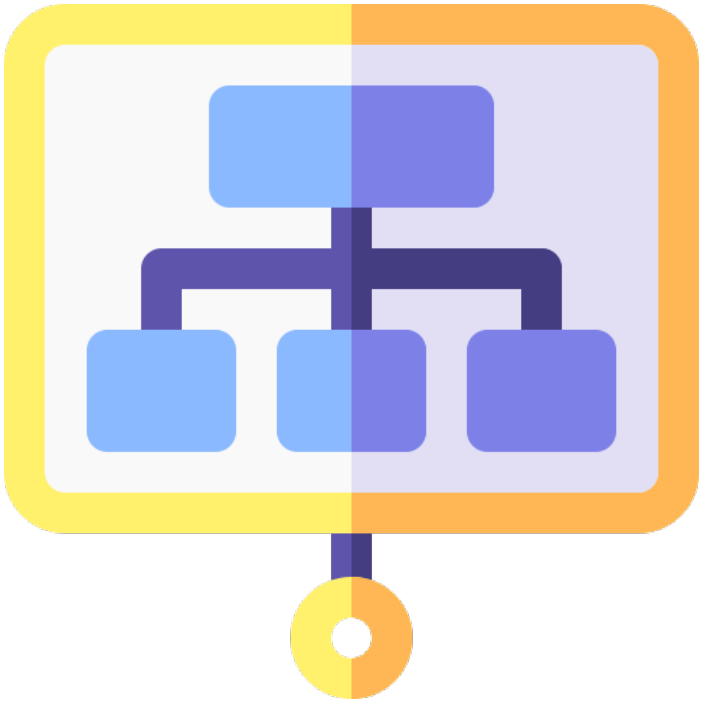
mode

factor

min_lr

گام ۱: تعریف

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    min_lr=0.001,
    verbose=2
)
```

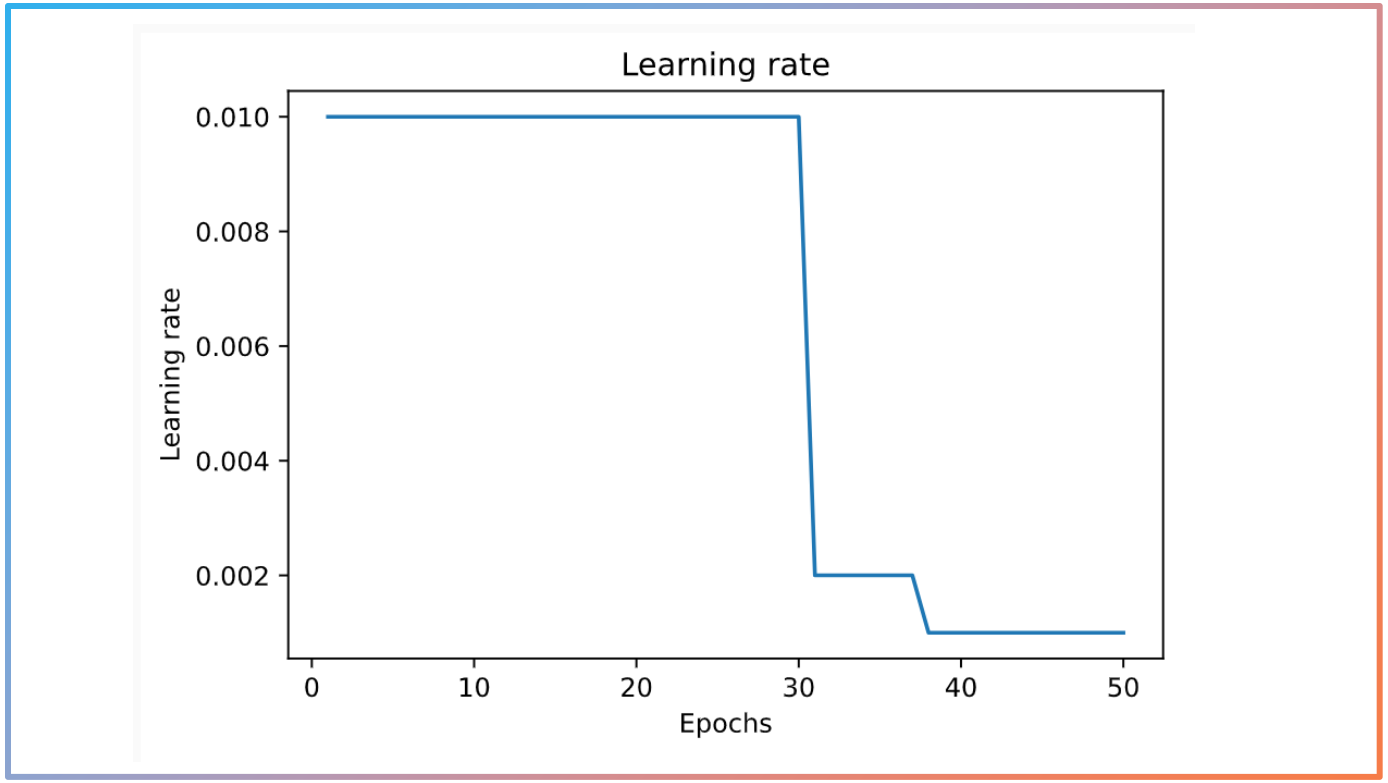


گام ۲ : استفاده

```
history_reduce_lr = model.fit(
    X_train,
    y_train,
    epochs=50,
    validation_split=0.20,
    batch_size=64,
    verbose=2,
    callbacks=[reduce_lr]
)
```

خروجی:

```
Epoch 00050: ReduceLROnPlateau reducing learning rate to 0.001. 8000/8000 - 1s - loss:
0.3696 - accuracy: 0.8742 - val_loss: 0.4483 - val_accuracy: 0.8455
```



LearningRateScheduler

گام ۱: تعریف

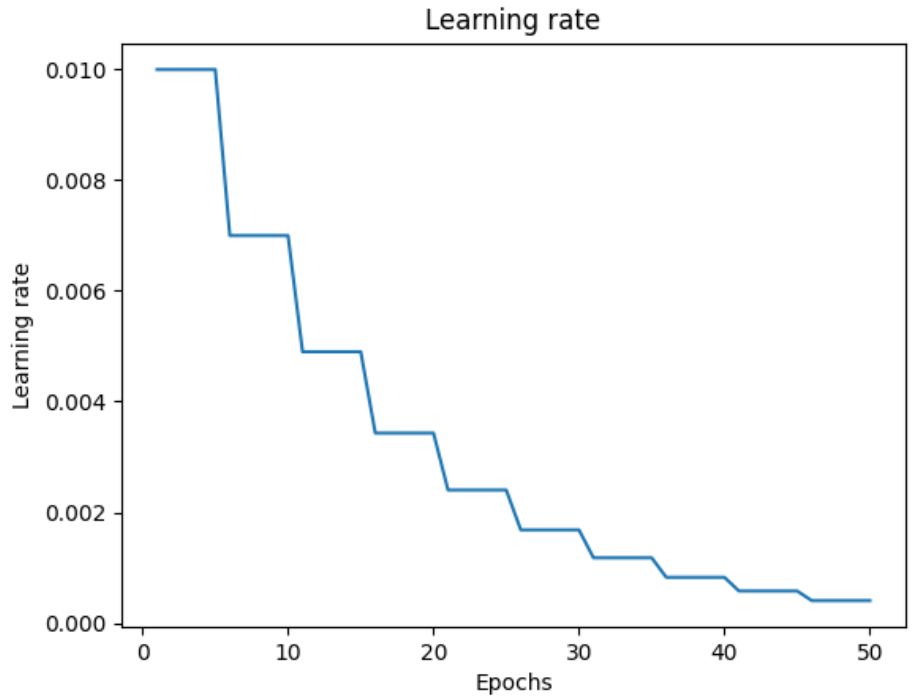
```
def lr_decay(epoch, lr):
    if epoch != 0 and epoch % 5 == 0:
        return lr * 0.7
    return lr
```

گام ۲ : استفاده

```
history_lr_schedule = model.fit(
    X_train,
    y_train,
    epochs=20,
    validation_split=0.20,
    batch_size=64,
    verbose=2,
    callbacks=[LearningRateScheduler(lr_decay, verbose=1)]
```

خروجی:

```
Epoch 00006: LearningRateScheduler setting learning rate to 0.006999999843537807.

125/125 - 0s - loss: 0.6128 - accuracy: 0.8027 - val_loss: 0.6335 - val_accuracy: 0.7815
```



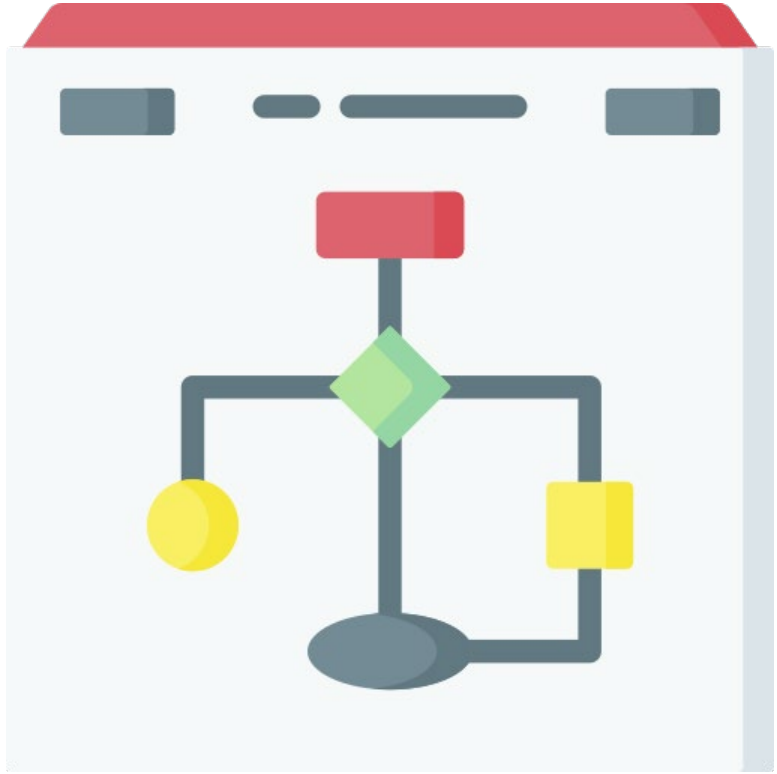
TensorBoard

به کمک این کالبک میتوانیم برای TensorBoard لاگ تولید کنیم.

راه اندازی و استفاده از TensorBoard

```
tf.keras.callbacks.TensorBoard(log_dir='logs')
```

```
tensorboard - -logdir = path_to_your_logs
```



TerminateOnNan

وقتی که مقدار Loss برابر Nan شد، فرایند Training را متوقف کن.

```
tf.keras.callbacks.TerminateOnNaN()
```

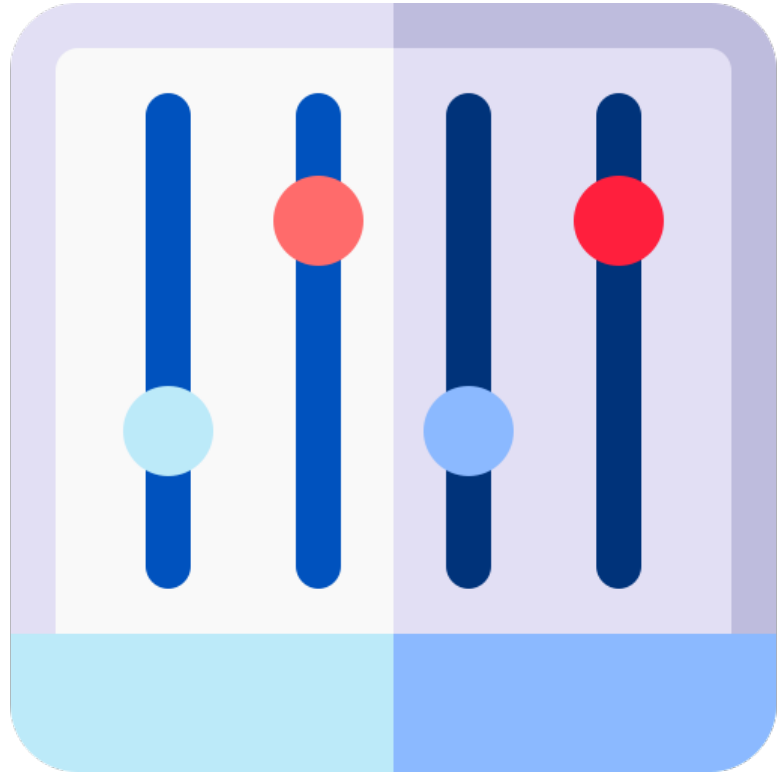
BaseLogger & History

این دو کالبک به صورت پیش فرض اعمال می شود و آنچه را که در طول فرآیند Train مشاهده میکنیم و خروجی History حاصل این دو کالبک است.

LambdaCallback

پارامترهای این کالبک :

```
tf.keras.callbacks.LambdaCallback(
    on_epoch_begin=None,
    on_epoch_end=None,
    on_batch_begin=None,
    on_batch_end=None,
    on_train_begin=None,
    on_train_end=None,
)
```



ورودی های متناظر:

```
on_epoch_begin and on_epoch_end expect two positional arguments: epoch, logs
on_batch_begin and on_batch_end expect two positional arguments: batch, logs
on_train_begin and on_train_end expect one positional argument: logs
```

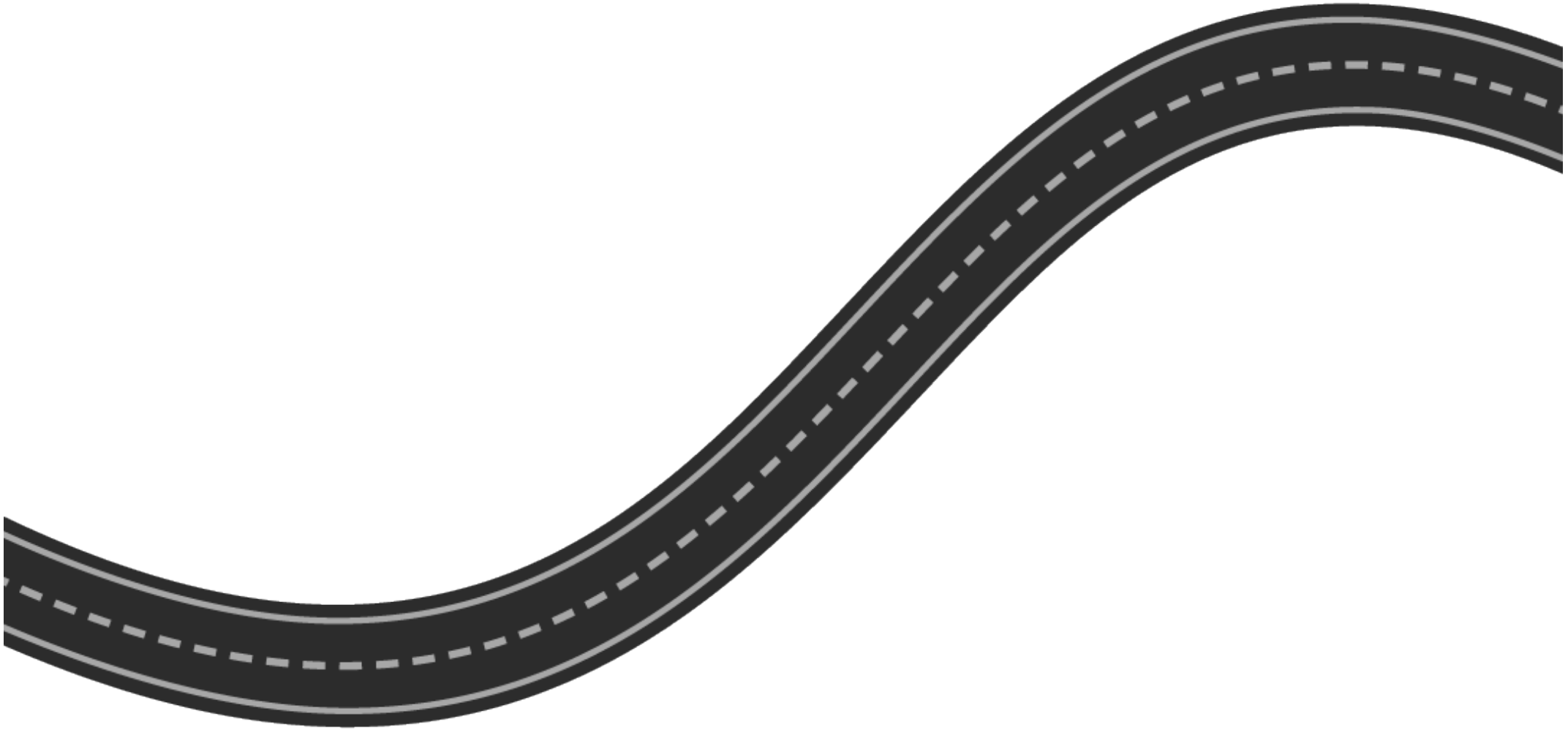
گام ۱: تعریف

```
epoch_callback = LambdaCallback(
    on_epoch_begin=lambda epoch,logs: print('Starting Epoch {}!'.format(epoch+1)))

batch_loss_callback = LambdaCallback(
    on_batch_end=lambda batch,logs: print('\n After batch {}, the loss is {:7.2f}.'.
format(batch, logs['loss'])))

train_finish_callback = LambdaCallback(
    on_train_end=lambda logs: print('Training finished!'))
```

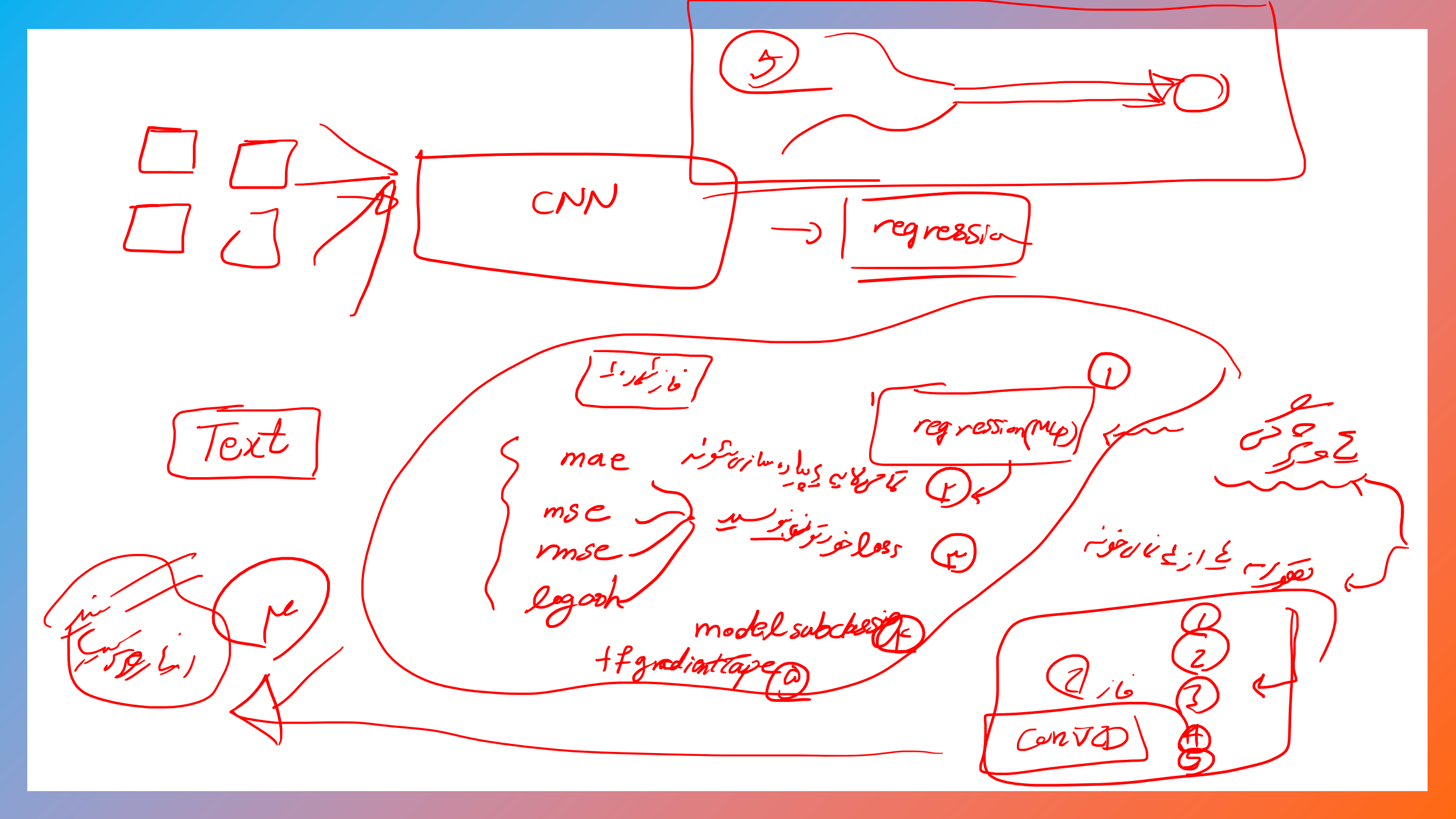
گام ۲ : استفاده

```
history_lambda_callback = model.fit(
    X_train,
    y_train,
    epochs=2,                    # change epoch to 2 for demo purpose
    validation_split=0.20,
    batch_size=2000,             # change to 2000 for demo purpose
    verbose=False,
    callbacks=[epoch_callback, batch_loss_callback, train_finish_callback]
)
```

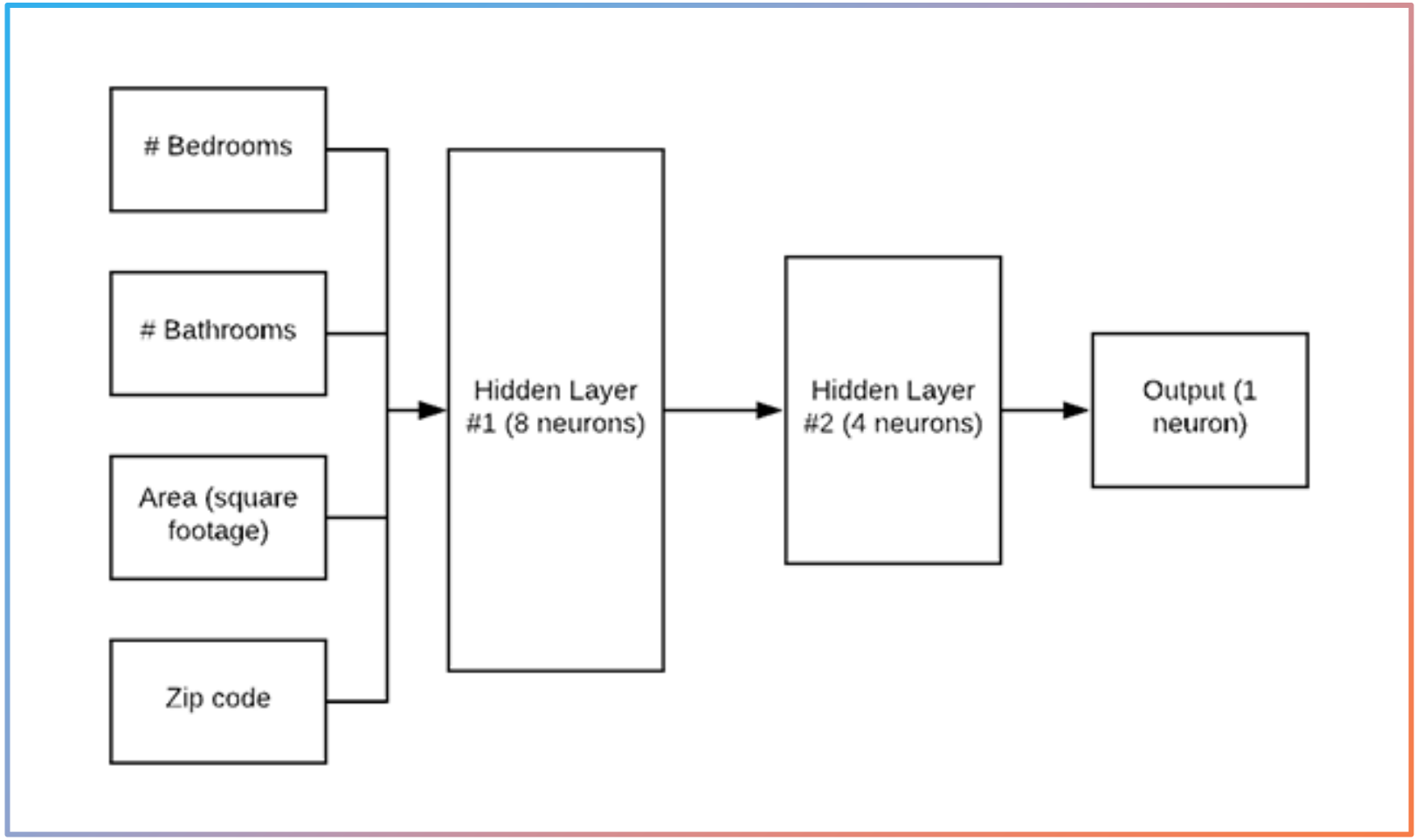
چه گفته ایم و چه خواهیم گفت:



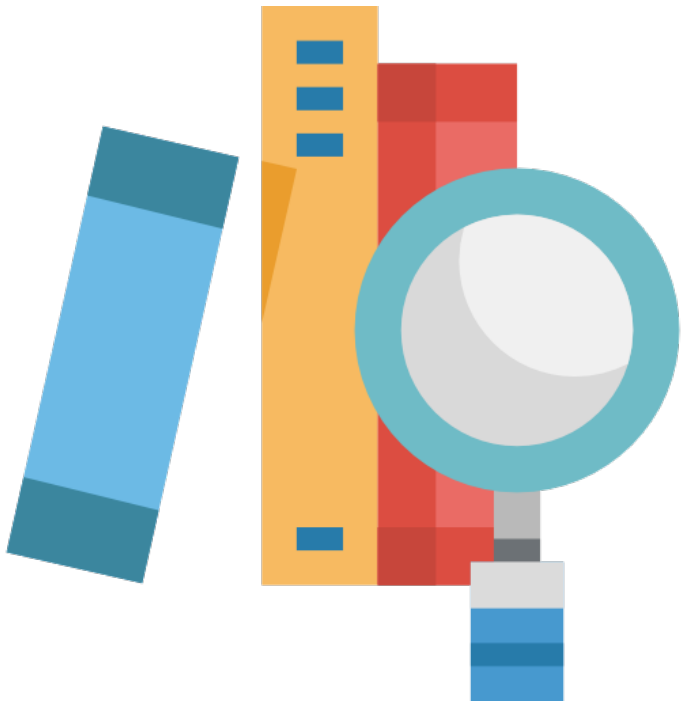
حل یک مثال کامل


CNN
regression
Text
regression(MLP)
mae
mse
rmse
logcosh
model subclassing
tf.gradientTape
Conv2D

ساختار شبکه عصبی



اضافه کردن کتابخانه های مورد نیاز



```
import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import LabelBinarizer

from sklearn.model_selection import train_test_split

from tensorflow.keras import layers, models

from tensorflow.keras.optimizers import Adam
```

استخراج کد منطقه یکتا

```
def load_house_attributes(inputPath):

    cols = ["bedrooms", "bathrooms", "area", "zipcode", "price"]

    df = pd.read_csv(inputPath, sep=" ", header=None, names=cols)

    zipcodes = df["zipcode"].value_counts().keys().tolist()

    counts = df["zipcode"].value_counts().tolist()
```

حذف مناطق با داده های کم

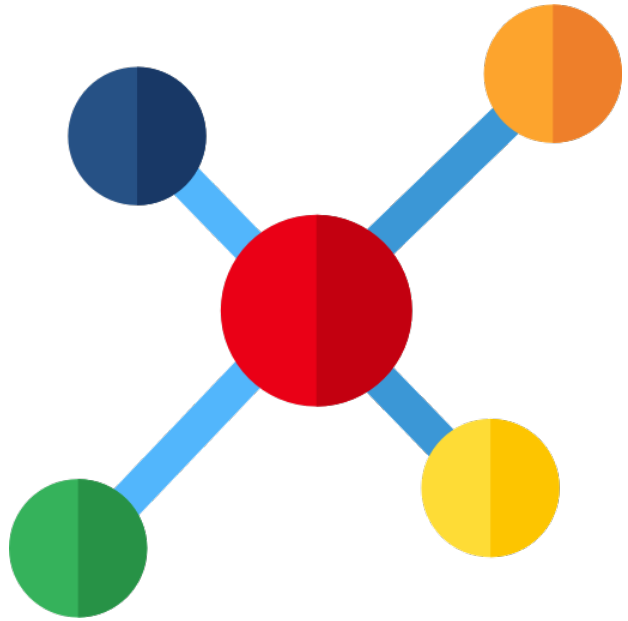
```
for (zipcode, count) in zip(zipcodes, counts):
    if count < 25:
        idxs = df[df["zipcode"] == zipcode].index
        df.drop(idxs, inplace=True)
return df
```

فراخوانی تابع و تبدیل به Train و Test

```
df = load_house_atrributes("HousesInfo.txt")

train, test = train_test_split(df, test_size=0.25, random_state=42)
```

نرمالایز کردن ویژگی های پیوسته

```
def preprocess_house_attribute(df, train, test):
    continuous = ["bedrooms", "bathrooms", "area"]
    sc = MinMaxScalar()
    trainContinuous = sc.fit_transform(train[continuous])
    testContinuous = sc.fit_transform(test[continuous])
```



One Hot کردن کد پستی

```
zipBinarizer = LabelBinarizer().fit(df["zipcode"])

trainCategorical = zipBinarizer.transform(train["zipcode"])

testCategorical = zipBinarizer.transform(test["zipcode"])

trainX = np.hstack([trainCategorical, trainContinuous])

testX = np.hstack([testCategorical, testContinuous])

return (trainX, testX)
```

تقسیم بندی داده ها و نرمالایز کردن قیمت ها

```
maxPrice = train["price"].max()

trainY = train["price"] / maxPrice

testY = test["price"] / maxPrice
```

ساخت شبکه عصبی

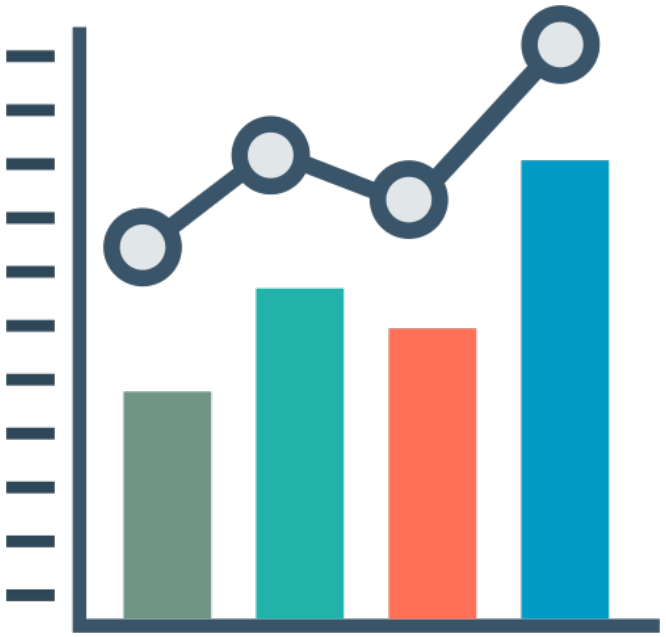
```
def create_mlp(dim):
    model = models.Sequential([
                                layers.Dense(8, input_dim = dim, activation="relu"),
                                layers.Dense(4, activation="relu"),
                                layers.Dense(1, activation="linear")
                                ])
    return model
```

تنظیم ویژگی ها و Train شبکه

```
model = create_mlp(trainX.shape[1])

opt = Adam(lr=1e-3)

model.compile(loss="mean_absolute_percentage_error", optimizer=opt)

model.fit(x=trainX, y=trainY,validation_data=(testX, testY),

          epochs=200, batch_size=8)
```

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

نمایش نتایج در خروجی



```
preds = model.predict(testX)

diff = preds.flatten() - testY

percentDiff = (diff / testY) * 100

absPercentDiff = np.abs(percentDiff)

mean = np.mean(absPercentDiff)

std = np.std(absPercentDiff)

print("[INFO] mean: {:.2f}%, std: {:.2f}%".format(mean, std))
```

تمرین : همه چیز را از صفر پیاده سازی کنید.